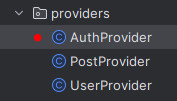
**ACTIVIDAD N° 7:**

**Clase AuthProvider**



La clase UserProvider interactúa con una base de datos basada en Parse Server para realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) relacionadas con un objeto de la clase User.

También podemos decir que la clase UserProvider actúa como un proveedor de datos que encapsula las operaciones necesarias para manejar usuarios en una base de datos Parse. Permite:

* Crear usuarios.
* Recuperar usuarios por correo electrónico.
* Actualizar información de usuarios.
* Eliminar usuarios por su ID.

Esto facilita la integración del backend con la lógica de negocio y la interfaz de usuario en una aplicación Android.

**MÉTODOS**

1. **Constructor: AuthProvider(Context context)**

public AuthProvider(Context context) {  
 Parse.initialize(new Parse.Configuration.Builder(context)  
 .applicationId(context.getString(R.string.back4app\_app\_id))  
 .clientKey(context.getString(R.string.back4app\_client\_key))  
 .server(context.getString(R.string.back4app\_server\_url))  
 .build()  
 );  
}

return result;  
}

Parse.initialize se utiliza para:

Parse.Configuration.Builder(context) se utiliza para construir la configuración Context

Valores de configuración :

* + applicationId:Este es el ID único de su aplicación.Back4App se
  + clientKey:Esta claveapplicationId, está disponible en tu
  + server:Esta es la URL parahttps://parseapi.back4app.com/

context.getString(R.string.xxx) estos métodos recuperan el applicationId, `clclientKey, y serverURL desde \*strings.xmlarchivo (de la carpeta values)

**2. public LiveData<String> signIn(String email, String password)**

Este método recibe el parámetro “email” y “password” de tipo String, para obtener un usuario en Parse. Este código de Android realiza una autenticación de usuario utilizando Parse, una plataforma de backend.

public LiveData<String> signIn(String email, String password) {  
 MutableLiveData<String> authResult = new MutableLiveData<>();  
 ParseUser.logInInBackground(email, password, new LogInCallback() {  
 @Override  
 public void done(ParseUser user, ParseException e) {  
 if (e == null) {  
 // Login exitoso  
 authResult.setValue(user.getObjectId());  
 Log.d("AuthProvider", "Usuario autenticado exitosamente: " + user.getObjectId());  
 } else {  
 // Error en el login  
 Log.e("AuthProvider", "Error en inicio de sesión: ", e);  
 authResult.setValue(null);  
 }  
 }  
 });  
 return authResult;  
}

**Método signIn** toma un correo electrónico y una contraseña como parámetros y devuelve un LiveData<String> que contiene el resultado de la autenticación.

**MutableLiveData<String> authResult**, crea un objeto MutableLiveData para almacenar el resultado de la autenticación.

**ParseUser.logInInBackground** este método intenta iniciar sesión con el correo electrónico y la contraseña proporcionados de manera asíncrona.

**Callback LogInCallback**, define un callback para manejar el resultado de la autenticación:

* + **Si la autenticación es exitosa (e == null)**:
    - Se establece el valor de authResult con el ID del usuario autenticado (user.getObjectId()).
    - Se registra un mensaje de éxito en los logs.
  + **Si hay un error en la autenticación**:
    - Se registra el error en los logs.
    - Se establece el valor de authResult como null.

**Retorno de authResult**: Finalmente, el método devuelve el LiveData que contiene el resultado de la autenticación.

La interfaz de usuario observa el LiveData para reaccionar a los cambios en el estado de autenticación de manera reactiva.

**3. signUp(String email, String password)**

Este método recibe el parámetro “email” y “password” de tipo String, luego realiza el registro de un nuevo usuario utilizando Parse

// Registro con Parse  
public LiveData<String> signUp(String email, String password) {  
 MutableLiveData<String> authResult = new MutableLiveData<>();  
 ParseUser user = new ParseUser();  
 user.setUsername(email);  
 user.setPassword(password);  
 user.signUpInBackground(e -> {  
 if (e == null) {  
 // Registro exitoso  
 authResult.setValue(user.getObjectId());  
 Log.d("AuthProvider", "Usuario registrado exitosamente: " + user.getObjectId());  
 } else {  
 // Error en el registro  
 Log.e("AuthProvider", "Error en registro: ", e);  
 authResult.setValue(null);  
 }  
 });  
 return authResult;  
}

**Método signUp** toma un correo electrónico y una contraseña como parámetros y devuelve un LiveData<String> que contiene el resultado del registro.

**MutableLiveData<String> authResult** se crea un objeto MutableLiveData para almacenar el resultado del registro.

**Creación de ParseUser** se crea una nueva instancia de ParseUser y se configuran el nombre de usuario y la contraseña con los valores proporcionados.

**user.signUpInBackground** este método intenta registrar al usuario de manera asíncrona.

**Callback de registro el**  callback maneja el resultado del registro:

**Si el registro es exitoso (e == null)**:

Se establece el valor de authResult con el ID del usuario registrado (user.getObjectId()).

Se registra un mensaje de éxito en los logs.

**Si hay un error en el registro**:

Se registra el error en los logs.

Se establece el valor de authResult como null.

**Retorno de authResult**: Finalmente, el método devuelve el LiveData que contiene el resultado del registro.

La interfaz de usuario observa el LiveData para reaccionar a los cambios en el estado de autenticación de manera reactiva.

**4. public LiveData<String> getCurrentUserID()**

Este método no recibe parámetros, su tarea es obtener el ID del usuario actualmente autenticado utilizando Parse

public LiveData<String> getCurrentUserID() {  
 MutableLiveData<String> currentUserId = new MutableLiveData<>();  
 ParseUser currentUser = ParseUser.getCurrentUser();  
 if (currentUser != null) {  
 currentUserId.setValue(currentUser.getObjectId());  
 }  
 return currentUserId;  
}

**Método getCurrentUserID** devuelve un LiveData<String> que contiene el ID del usuario actualmente autenticado.

**MutableLiveData<String> currentUserId** se crea un objeto MutableLiveData para almacenar el ID del usuario actual.

**ParseUser.getCurrentUser()** este método obtiene el usuario actualmente autenticado.

**Verificación del usuario actual**:

* **Si el usuario actual no es null**: establece el valor de currentUserId con el ID del usuario actual (currentUser.getObjectId()).

**Retorno de currentUserId**: Finalmente, el método devuelve el LiveData que contiene el ID del usuario actual.

La interfaz de usuario observa el LiveData para reaccionar a los cambios en el estado de autenticación de manera reactiva.

**5. public LiveData<Boolean> logout()**

Este método es para la desconexión de un usuario utilizando Parse.

MutableLiveData<Boolean> logoutResult = new MutableLiveData<>();  
 ParseUser.logOutInBackground(e -> {  
 if (e == null) {  
 logoutResult.setValue(true);  
 if (getApplicationContext() != null) {  
 getApplicationContext().getCacheDir().delete();  
 }  
 Log.d("AuthProvider", "Caché eliminada y usuario desconectado.");  
  
 } else {  
  
 logoutResult.setValue(false);  
 Log.e("AuthProvider", "Error al desconectar al usuario: ", e);  
 }  
 });  
 return logoutResult;  
}

**MutableLiveData<Boolean> logoutResult** se crea un objeto MutableLiveData para almacenar el resultado de la desconexión.

**ParseUser.logOutInBackground** intenta desconectar al usuario de manera asíncrona.

**Callback de desconexión** el callback es para manejar el resultado de la desconexión:

**Si la desconexión es exitosa (e == null)**:

* Se establece el valor de logoutResult como true.
* Si el contexto de la aplicación no es null, se elimina la caché de la aplicación (getApplicationContext().getCacheDir().delete()).
* Se registra un mensaje de éxito en los logs.

**Si hay un error en la desconexión**:

* Se establece el valor de logoutResult como false.
* Se registra el error en los logs.

**Retorno de logoutResult**: Finalmente, el método devuelve el LiveData que contiene el resultado de la desconexión.

**CÓDIGO**

import static com.parse.Parse.getApplicationContext;  
  
import com.example.appchat.R;  
import com.parse.Parse;  
import com.parse.ParseUser;  
import com.parse.ParseException;  
import android.content.Context;  
import android.util.Log;  
import androidx.lifecycle.LiveData;  
import androidx.lifecycle.MutableLiveData;  
import com.parse.LogInCallback;

public class AuthProvider {  
 public AuthProvider(Context context) {  
 Parse.initialize(new Parse.Configuration.Builder(context)  
 .applicationId(context.getString(R.string.back4app\_app\_id))  
 .clientKey(context.getString(R.string.back4app\_client\_key))  
 .server(context.getString(R.string.back4app\_server\_url))  
 .build()  
 );  
 }  
  
 public LiveData<String> signIn(String email, String password) {  
 MutableLiveData<String> authResult = new MutableLiveData<>();  
 ParseUser.logInInBackground(email, password, new LogInCallback() {  
 @Override  
 public void done(ParseUser user, ParseException e) {  
 if (e == null) {  
 // Login exitoso  
 authResult.setValue(user.getObjectId());  
 Log.d("AuthProvider", "Usuario autenticado exitosamente: " + user.getObjectId());  
 } else {  
 // Error en el login  
 Log.e("AuthProvider", "Error en inicio de sesión: ", e);  
 authResult.setValue(null);  
 }  
 }  
 });  
 return authResult;  
 }  
  
 // Registro con Parse  
 public LiveData<String> signUp(String email, String password) {  
 MutableLiveData<String> authResult = new MutableLiveData<>();  
 ParseUser user = new ParseUser();  
 user.setUsername(email);  
 user.setPassword(password);  
 user.signUpInBackground(e -> {  
 if (e == null) {  
 // Registro exitoso  
 authResult.setValue(user.getObjectId());  
 Log.d("AuthProvider", "Usuario registrado exitosamente: " + user.getObjectId());  
 } else {  
 // Error en el registro  
 Log.e("AuthProvider", "Error en registro: ", e);  
 authResult.setValue(null);  
 }  
 });  
 return authResult;  
 }  
  
 public LiveData<String> getCurrentUserID() {  
 MutableLiveData<String> currentUserId = new MutableLiveData<>();  
 ParseUser currentUser = ParseUser.getCurrentUser();  
 if (currentUser != null) {  
 currentUserId.setValue(currentUser.getObjectId());  
 }  
 return currentUserId;  
 }  
  
 public LiveData<Boolean> logout() {  
 MutableLiveData<Boolean> logoutResult = new MutableLiveData<>();  
 ParseUser.logOutInBackground(e -> {  
 if (e == null) {  
 logoutResult.setValue(true);  
 if (getApplicationContext() != null) {  
 getApplicationContext().getCacheDir().delete();  
 }  
 Log.d("AuthProvider", "Caché eliminada y usuario desconectado.");  
  
 } else {  
  
 logoutResult.setValue(false);  
 Log.e("AuthProvider", "Error al desconectar al usuario: ", e);  
 }  
 });  
 return logoutResult;  
 }  
}